METHOD FOR ACCELERATED PACKET PROCESSING

Field of the Invention

[0001] This invention relates to communication systems and in particular to a packet processing scheme in which a cache is used to quickly access upper-layer headers in packet flows.

Background of the Invention

[0002] In the following discussion specific reference is made to Internet protocol version 6 (IPv6) and multi-field classification (MFC). It is to be understood, however, that the concepts of the present invention are not limited to IPv6 and MFC but the described implementation is intended as exemplary only.

[0003] IPv6 represents an evolutionary development to overcome some limitations in IPv4. In particular, the routing and addressing limitations of IPv4 place a real limit on the address configurations available based on a 32 bit field limit. IPv6 has increased the address field to 128 bits. This supports more levels of addressing hierarchy as well as a much greater number of addressable nodes. Additionally, some IPv4 header fields have been dropped to reduce the processing cost of packet handling and to limit the bandwidth cost of the IPv6 header. A new capability has been added to IPv6 which enables labeling of packets belonging to particular traffic flows for which the sender requests special handling based on quality of service or real time handling. Additionally, the mechanism to add optional internet-layer information to a packet has changed. In IPv4, this is achieved through the option field which is a part of the IPv4 header. The Internet Header Length (IHL) field in the IPv4 header indicates the length of these extra fields and the basic header. However, due to the length of the IHL field, only a limited number of options could be added to a packet. In IPv6, internet-layer information is added using extension headers. These headers are added to an IPv6 packet between the IPv6

header and the upper-layer header. Each extension header contains a field identifying the type of the next header, and instead of using a field similar to the IHL of IPv4, IPv6 specifies the length of each extension header in the extension headers themselves. This allows an unlimited number of extension headers to be added to an IPv6 packet.

[0004] Figure 1 shows the IPv6 header format including the flow label field.

[0005] Regular packet processing typically requires fields from upper-layer headers. Multi-Field Classification (MFC), one of many examples of processing such data, usually classifies packets based on the 5-tuple <Source IP, Destination IP, Source Port, Destination Port, Protocol>. When options or extension headers are used, it can become difficult and costly to reach the upper-layer header in order to retrieve the data necessary for packet processing.

[0006] For IPv4, processing packets with options is simple as the aforementioned IHL field can be used to skip over the entire IPv4 header, including any appended options, to reach the upper-layer header.

[0007] For IPv6, reaching the upper-layer header is significantly more difficult for packets containing extension headers as the IPv6 header does not have a field like the IHL in IPv4 that can be used to immediately reach the upper-layer header. One prior art solution is to traverse the extension headers serially. Since the length of each extension header is stored in the extension header itself, this appears to be the most obvious solution. Thus, IPv6 packet processing becomes costly if a number of extension header need to be traversed before the upper-layer header can be reached and regular packet processing performed. This cost seems even more unnecessary since the presence of IPv6 extension headers does not necessarily indicate that special handling is required from the node. Some extension headers

contain information relevant only to the destination node, and can be completely ignored by all nodes en-route to the destination. In fact, only hop-by-hop and router extension headers are relevant to all nodes and the latter is only applicable when the arriving packet is destined for the node. Fortunately, RFC2460 specifies that hop-by-hop extension headers be the first extension header after the IPv6 header. Thus a node can quickly determine if special handling for a packet is required simply by examining the destination address and the Next Header field of the IPv6 header.

[0008] An alternative to the previous approach is to simply limit the fields used in packet classification to those from the IP header such that the upper-layer header does not need to be examined. For MFC, this reduces the standard 5-tuple classification to a 3-tuple classification. This is not a very popular option as limiting the fields used in the MFC key for IPv6 means changing the way security of the node is controlled. Thus, a new set of rules must then be developed for IPv6 simply because it is sometimes inconvenient to read all the fields from the upper-layer header. Because of this, limiting the fields used in packet classification is not considered a competing solution.

Summary of the Invention

[0009] Simply stated, the present invention relates to the caching of information about the length of extension headers used within a packet flow in order to accelerate the handling of subsequent packets in the flow. In an exemplary embodiment the packet flow comprises IPv6 packets.

[0010] Therefore in accordance with a first aspect of the present invention there is provided a method of accessing upper-layer headers in a packet flow, comprising the steps of: responsive to a packet containing extension headers, building a cache key based on the fields present in the header and performing a cache lookup for a

cache entry; and responsive to finding a corresponding cache entry, reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

[0011] In accordance with a second aspect of the present invention there is provided a system for performing MFC for filtering a packet flow, comprising: means responsive to an IPv6 packet header containing extension headers, for building a cache key based on the fields present in the header and performing a cache lookup for a cache entry; and means responsive to finding a corresponding cache entry, for reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

Brief Description of the Drawings

[0012] The invention will now be described in greater detail with reference to the attached drawings wherein:

[0013] Figure 1 illustrates the IPv6 header format;

[0014] Figure 2 illustrates the sequence of operations needed to reach the upper-layer header of an IPv6 packet;

[0015] Figures 3A and 3B show a comparison of memory accesses using serial and plural processing and

[0016] Figures 4A to 4G illustrate time lines of extension header lookups using options.

Detailed Description of the Invention

[0017] When a node receives a packet, the IPv6 header will be examined. Packets which have no extension headers will be handled using regular packet handling techniques. This means that the only additional cost for processing these packets is the check for extension headers. For packets with the hop-by-hop extension header, or packets destined to the node containing routing headers, the special handling specified by the header can be launched at this point.

[0018] For packets with extension headers the node will use the information contained in the IPv6 header to build a key in order to perform a cache lookup. The format of this key could be the same for all packets, or could be based on the presence or value of fields in the IPv6 header. For example, for packets with non-zero flow labels the tuple <IPv6.srcIP, IPv6.flowLabel>, or <IPv6.srcIP, IPv6.flowLabel, IPv6.nextHeader> could be used. For packets with a zero flow label the tuple <IPv6.srcIP, IPv6. dstIP> or <IPv6.srcIP, IPv6. dstIP, IPv6.nextHeader> could be used.

[0019] While the cache lookup is executing, the information from the header can be used to read the first extension header in the list. If no entry is found in the cache the node proceeds to traverse the list of extension headers serially using the extension header length and next header fields of the extension headers. After a serial traversal, the data used to traverse the extension headers is placed in the cache in anticipation of more packets in the same flow with similar extension headers. However, if a cached entry is found, the node uses the information from the first extension header and the cached data to read each of the remaining extension headers in parallel. This allows the node to quickly traverse the extension headers and it will either arrive at the upper-layer header, or, in the event that not enough data was cached, continue with a serial traversal of the final extension headers. If the cached data is found to be incorrect at any point in the

parallel traversal of the extension headers, the node must traverse the remaining extension headers serially from the point at which the data was incorrect. In this case the cache would be updated to reflect the extension headers in the current packet in anticipation of additional packets in the same flow with similar format.

[0020] Figure 2 summarizes the algorithm described above. It should be noted that, if after reading the IPv6 header it is known that there are no extension headers in the packet then no cache lookup is performed. Thus, in cases where there are no packets using extension headers there will be only a small, if any, penalty for implementing these optimizations.

[0021] Figure 3 shows the difference in the memory accesses between traversing the extension header linked list and using the accelerated algorithm. In the case shown, the packet processing latency is reduced by one memory access. Figure 3A shows the memory accesses used in a serial traversal of the header. As shown in Figure 3B the parallel processing involves at step 2B an examination of the cache in parallel with the review of the first extension header.

[0022] There are several alternate embodiments to the invention. The first embodiment is the ability to identify, and react to "unpredictable" flows. This would be used to detect flows in which the extension headers were constantly changing. If such a flow were identified, it would be clear that the cached data would likely be incorrect, and that it would be more efficient to traverse the extension headers serially, instead of attempting a parallel traversal. Additionally, for such a flow, updating the cache to reflect the format of the current packet would be unnecessary. Marking a flow as "unpredictable" could be the result of manual configuration or determined dynamically based on observing the flow.

[0023] The second embodiment is to have the ability to specify a cache entry as "sticky". This would indicate that the majority of the packets in a flow should be correctly interpreted using the cached data, and that in the event that the cached data does not match the packet the cache should not be updated. This would prevent a single packet from changing the cached data. As in the previous extension, this feature could be manually enabled or enabled as a result of flow observations. This extension could be enhanced by specifying that in the case of cache failures, the data in the cache would only be updated after a certain number of failures.

[0024] A third embodiment is to also cache some of the upper-layer header information such as the protocol and the source and destination ports. When the cached information is received, the protocol information, combined with the information contained in the IPv6 header, can be used to immediately begin packet classification, such as MFC. While the classification is completing, the remaining cached information can be used to quickly traverse the extension headers and to verify that the cached protocol information matches that of the packet. If the two sets of information match then the classification is valid. If, however, the information from the cache is incorrect, classification needs to be repeated with the correct information from the upper-layer header. In the worst case, there is a cost of one additional classification, but in the best case this allows for packet classification to be done in parallel with the extension header traversal and can reduce the total packet processing latency by the latency required for packet classification.

[0025] A still further embodiment is the ability to cache the results of other classifications and lookups performed during regular packet processing. For example, the result of the MFC, and forwarding lookups could be added to the cache. Using this embodiment, only the first packet in a flow needs to be fully

classified as subsequent packets in the flow are able to use the results of this classification from the cache.

[0026] The present invention provides advantages over the previously discussed prior art solutions. As a large portion of extension headers require no special processing by most nodes, the present invention reduces the cost associated with processing IPv6 packets containing extension headers. This allows the upper-layer headers to be quickly reached such that packet processing can continue.

[0027] With respect to the use of existing MFC tuples, since the upper-layer header can be quickly located, the fields used in packet classification can remain unchanged with respect to those which were used for IPv4. This means that existing security policies can be applied to IPv6 in the same manner as they were in IPv4.

[0028] The cost of loading extension header information in parallel may results in a small increase in total memory bandwidth due to the need to read the data from the cache, but this increase will only occur for packets which contain extension headers.

[0029] Figure 4 shows the timelines for the parallel loading of up to six extension headers. The solid bars indicate the best case traversal of the extension headers. The dashed bars show the absolute worst case traversal of the extension headers in the event that the cached information is incorrect. It can be seen from Figures 4A to 4G that in the best case traversal of the extension headers considerable time is saved from the process involving serial header lookups as indicated by the dashed bars.

[0030] The cache can be implemented in any number of ways. Possible implementations include the use of a context addressable memory (CAM) or through the use of hash tables. Perhaps the most obvious method is to use an existing CAM. Using the CAM as a cache has its advantages and disadvantages. The main advantage is that with a CAM is that most, if not all, the bits from the key can be used to perform a lookup resulting in very low or no chance, depending on the exact implementation, for there to be a collision when matching a key. This means there will be a very few mismatches due to obtaining the wrong information from the CAM. The downside to a CAM implementation, apart from the fact that CAMs are expensive and constrained by space, is that the CAM lookup may require two memory accesses to do a cache lookup. The first access, a write, provides the CAM with the key and the instruction to perform a lookup. The second access, a read, retrieves the result from the CAM.

[0031] As an alternative to using the CAM a hash table can be used to implement the cache. This implementation has certain advantages since it does not consume precious and costly CAM space. Due to the speed that a hash lookup can be performed, this implementation is likely at least as fast, and certainly less costly, as the CAM implementation. The downside to the hash implementation is that there is a greater probability for a collision when matching a key compared to the CAM implementation.

[0032] Depending on exactly how the cache is implemented, it may be possible for a collision to occur when matching a key. This is the result of not using all of the bits of the key as an index into the cache. The effect is that further processing may be required to resolve this collision. It may be that in certain implementations, the cost of resolving these collisions is greater than the cost of serially traversing the extension headers due to incorrect cache information. In these cases, it would be beneficial to simply assume that collisions do not occur. The end result would be

an increase in the frequency of cache and packet data mismatch, but an overall reduction in processing. For hash implementations, this has the additional benefit of reducing the size of the hash table entries as the information that was needed to resolve collisions is no longer needed. The reduction in hash table entry size effectively increases the number of entries in the table, or allows for memory savings.

[0033] As discussed previously, the caching method uses fields from the header in order to build a key to index into the cache. The information stored in the cache is a duplication of the information believed to be in the packet. Because of this duplication, care must be taken to ensure that the cached data matches the data in the packet. In order to do this, at a minimum the lengths of the extension headers must be cached so that the extension headers can be loaded to confirm they match the cached data. If this validation were not performed it would be possible for a malicious host to establish a valid flow, but modify subsequent packets in the flow in order to bypass some security mechanism. As an example, assume an implementation that simply caches the total length of all extension headers. When the first packet, containing many extension headers, in the flow arrives the extension headers are traversed serially, and an entry is added to the cache. MFC is performed on the packet, and the packet is accepted. Another packet in the flow arrives, and the cached data is read. The total extension header offset is used to read what is believed to be the upper-layer header, but which actually a fake header identical to that of the first packet. MFC is performed, and the packet is accepted. Had the extension headers been read serially it would have been clear that this second packet had fewer extension headers that the first packet, and that the real upper-layer header was located earlier in the packet than the one pointed to by the cached data.

[0034] Although particular embodiments of the invention can be described and illustrated it will be apparent to one skilled in the art that numerous changes can be made without departing from the basic concept of the invention. It is to be understood, however, that such changes will fall within the full scope of the invention as defined by the appended claims.